Solving 2D-Pattern Matching with Networks of Picture Processors

Henning Bordihn , Paolo Bottoni
Anna Labella , and Victor Mitrana

Abstract. We propose a solution based on networks of picture processors to the problem of picture pattern matching. The network solving the problem can be informally described as follows: it consists of two subnetworks, one of them extracts simultaneously all subpictures of the same size from the input picture and sends them to the second subnetwork. The second subnetwork checks whether any of the received pictures is identical to the pattern. We present an efficient solution based on networks with evolutionary processors only, for patterns with at most three rows or columns. Afterwards, we present a solution based on networks containing both evolutionary and hiding processors running in $\mathcal{O}(n+m+kl+k)$ computational (processing and communication) steps, where the input picture and the pattern are of size (n,m) and (k,l), respectively.

1 Introduction

Picture languages defined by different mechanisms have been studied extensively in the literature. Two-dimensional matrix and array models describing pictures have been proposed in [15,16,19,17]. On the other hand, models defining pictures that are connected arrays, but not necessarily rectangular, have been proposed as early as 70's [14] and a hierarchy of these grammars was considered in [18]. A new model of recognizable picture languages, extending to two dimensions the characterization of the one-dimensional recognizable languages in terms of alphabetic morphisms of local languages, has been introduced in [7]. Similarly to the string case, characterizations of recognizable picture series were proposed, see, e.g. [5,12]. An early survey on automata recognizing rectangular picture languages is [8], a bit more recent one considering different mechanisms defining

picture languages, not necessarily rectangular, is [14] and an even more recent and concise one is [6].

This work is a continuation of [4], where networks of evolutionary picture processors acting on rectangular pictures as *acceptors* are considered. The paper [4] is a first attempt to extend the investigation started in [10], where the data is organized in the form of linear strings, and continued in a series of papers (see [9] for a recent survey) to rectangular pictures. In [4], networks of evolutionary picture processors where each node is either a row/column substitution node or a row/column deletion node are considered. The action of each node on the data it contains is precisely defined. For instance, if a node is a row substitution node, then it can substitute a letter by another letter in either the top row only, the bottom row only, or an arbitrary row. Moreover, if there are more occurrences of the letter to be substituted in the row on which the substitution rule acts, then each such occurrence is substituted in different copies of that picture. An implicit assumption is that arbitrarily many copies of every picture are available. A similar informal explanation concerns the column substitution and deletion nodes. Local data is then transmitted over the network following a given protocol. Only data which can pass a filtering process can be communicated. This filtering process is regulated by input and output filters (defined by some very simple context conditions) associated with each node. All the nodes simultaneously send their data to, and receive data from, the nodes they are connected to. In [4] we showed that these networks can accept the complement of any local language, as well as languages that are not recognizable.

We consider here the pattern matching problem, which is largely motivated by different aspects in low-level image processing [13], and try to solve it in a parallel and distributed way with networks of picture processors. The network solving the problem can be informally described as follows: it consists of two subnetworks, one of them extracts simultaneously all subpictures of the same size from the input picture and sends them to the second subnetwork. In its turn, the second subnetwork consists of two subnetworks; one of them checks whether any of the received pictures is identical to the pattern, while the other one halts the computation if none of the received pictures is identical to the pattern. If the pattern is of size (k, l), with $1 \leq k \leq 3$, and $l \geq 1$, we present an efficient solution running in $\mathcal{O}(n+m+l)$ computational (processing and communication) steps, provided that the input picture is of size (n, m). Moreover, this solution can be extended at no further cost w.r.t. the number of computational steps to any finite set of patterns all of them of the same size.

We introduce a new operation and its inverse that can convert a visible row/column into an invisible one and vice versa. The two operations which seem to be relevant with respect to picture processing (see, e.g. "zoom-in", "zoom-out") are called *mask* and *unmask*, respectively. We show how this variant of networks of picture processors is able to solve efficiently (in $\mathcal{O}(n + m + kl + k)$ computational steps) the problem of pattern matching of an arbitrary pattern of size (k, l) in a given rectangular picture of size (n, m). Again, the solution can

be extended at no further cost w.r.t. the number of computational steps to any finite set of patterns all of them of the same size.

2 Basic Definitions

The basic terminology and notations concerning two-dimensional languages are taken from [6]. The set of natural numbers from 1 to n is denoted by $[n]$. The set of all finite subsets of a set A is denoted by 2^A. The cardinality of a finite set A is denoted by $card(A)$. Let V be an alphabet, V^* the set of one-dimensional strings over V and ε the empty string. A *picture* (or a two-dimensional string) over the alphabet V is a two-dimensional array of elements from V. We denote the set of all pictures over the alphabet V by V_*^*, while the empty picture will be still denoted by ε. A two-dimensional language over V is a subset of V_*^*.

Let π be a picture in V_*^*; we denote the number of rows and the number of columns of π by $\overline{\pi}$ and $|\pi|$, respectively. The pair $(\overline{\pi}, |\pi|)$ is called *the size* of the picture π. The size of the empty picture ε is obviously (n, m) with $nm = 0$. The set of all pictures of size (m, n) over the alphabet V, where $m, n \geq 1$, is denoted by V_m^n. The symbol placed at the intersection of the ith row with the jth column of the picture π, is denoted by $\pi(i, j)$.

Let π be a picture of size (m, n) over V; for any $1 \leq i \leq k \leq m$ and $1 \leq j \leq l \leq n$ we denote by ${}^{[i,j]}\pi_{[k,l]}$ the *subpicture* of π having its left-hand upper corner in $\pi(i, j)$ and right-hand lower corner in $\pi(k, l)$ (it starts and ends at (i, j) and (k, l) in π, respectively). For any values $i > k$ or $j > l$, we set ${}^{[i,j]}\pi_{[k,l]} = \varepsilon$. Furthermore, we simply write π instead of ${}^{[1,1]}\pi_{[m,n]}$.

For any alphabet V and a symbol $a \in V$, we denote by $\not{a}$ the *invisible* copy of a; furthermore, we set $\not{V} := \{\not{a} \mid a \in V\}$. We say that a picture $\pi \in (V \cup \not{V})_m^n$ is *well defined* if there exists $1 \leq i \leq k \leq m$ and $1 \leq j \leq l \leq n$ such that all elements of ${}^{[i,j]}\pi_{[k,l]}$ are from V and all the other elements of π are from $\not{V}$. In this case, we say that ${}^{[i,j]}\pi_{[k,l]}$ is the *maximal visible subpicture* of π. A rather intuitive way to understand a well defined picture π is to consider that some rows and/or columns of π are hidden but not deleted. Note that any picture over V is a well defined picture. For the rest of this paper, we deal with well defined pictures only. The minimal alphabet containing all visible symbols appearing in a picture π is denoted by $alph(\pi)$.

Let V be an alphabet; a rule of the form $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is called an *evolutionary rule*. We say that a rule $a \to b$ is: a) a *substitution rule* if both a and b are not ε; b) a *deletion rule* if $a \neq \varepsilon$, $b = \varepsilon$; c) an *insertion rule* if $a = \varepsilon$, $b \neq \varepsilon$. In this paper we shall ignore insertion rules because we want to process every given picture in a space bounded by the size of that picture. We denote the sets of substitutions and deletions by $Sub_V = \{a \to b \mid a, b \in V\}$ and $Del_V = \{a \to \varepsilon \mid a \in V\}$, respectively. Given a rule σ as above and a picture $\pi \in (V \cup \not{V})_m^n$, we define the following *actions* of σ on π following [4].

If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^{\leftarrow}(\pi)$ is the set of all pictures π' such that the following conditions are satisfied:

(1.) There exist $1 \leq u \leq v \leq m$ and $1 \leq s \leq t \leq n$ such that ${}^{[u,s]}\pi_{[v,t]}$ is the maximal visible subpicture of π,
(2.a.) There exists $u \leq i \leq v$ such that $\pi(i,s) = a$; then $\pi'(i,s) = b$, and $\pi'(j,l) = \pi(j,l)$ for all $(j,l) \in ([m] \times [n]) \setminus \{(i,s)\}$.
(2.b.) If the leftmostcolumn of ${}^{[u,s]}\pi_{[v,t]}$ does not contain any occurrence of a, then $\sigma^{\leftarrow}(\pi) = \{\pi\}$.
Informally, $\sigma^{\leftarrow}(\pi)$ is the set of all pictures that can be obtained from π by replacing an occurrence of a by b in the leftmost column of the maximal visible subpicture of π. Note that σ is applied to all occurrences of the letter a in the leftmost column of the maximal visible subpicture of π in different copies of the picture π. We say that the rule σ is applied to the leftmost column of the maximal visible subpicture of π.

In an analogous way, we define $\sigma^{\rightarrow}(\pi)$, $\sigma^{\uparrow}(\pi)$, $\sigma^{\downarrow}(\pi)$, $\sigma^{+}(\pi)$, as the set of all pictures obtained by applying σ to the rightmost column, to the first row, to the last row, and to any column/row of the maximal visible subpicture of π.

If $\sigma \equiv a \rightarrow \varepsilon \in Del_V$, then $\sigma^{\leftarrow}(\pi)$ is the picture obtained from π by deleting the i^{th} column of π provided that the maximal visible subpicture of π starts at the position (i,j) in π, for some j, and the i^{th} column of π contains an occurrence of a. If the leftmost column of the maximal visible subpicture of π does not contain any occurrence of a, then $\sigma^{\leftarrow}(\pi) = \pi$. We say that the deletion rule σ is applied to the leftmost column of the maximal visible subpicture of π.

Analogously, $\sigma^{\rightarrow}(\pi)$, $\sigma^{\uparrow}(\pi)$, and $\sigma^{\downarrow}(\pi)$ is the picture obtained from π by applying σ to the rightmost column, to the first row, and to the last row of the maximal visible subpicture of π, respectively. Furthermore, $\sigma^{|}(\pi)$ ($\sigma^{-}(\pi)$) is the set of pictures obtained from π by deleting an arbitrary column (row) containing an occurrence of a from π. If more than one column (row) of π contains a, then for each such column (row), there is a copy of π in $\sigma^{|}(\pi)$ ($\sigma^{-}(\pi)$) having this column (row) deleted. If π does not contain any occurrence of a, then $\sigma^{|}(\pi) = \{\pi\}(\sigma^{-}(\pi) = \{\pi\})$.

For every rule σ, symbol $\alpha \in \{\leftarrow, \rightarrow, \uparrow, \downarrow, |, -, +\}$, and $L \subseteq (V \cup \not\boxplus)^*_*$, we define the *$\alpha$-action of σ on L* by $\sigma^{\alpha}(L) = \bigcup_{\pi \in L} \sigma^{\alpha}(\pi)$. Given a finite set of rules M, we define the *α-action of M* on the picture π and the language L by:

$$M^{\alpha}(\pi) = \bigcup_{\sigma \in M} \sigma^{\alpha}(\pi) \quad \text{and} \quad M^{\alpha}(L) = \bigcup_{\pi \in L} M^{\alpha}(\pi),$$

respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary picture operations* since they may be viewed as the 2-dimensional linguistic formulations of local gene mutations.

We now define a new operation on pictures and its inverse, namely *mask* and *unmask*. Let π be a picture of size (m,n) over $V \cup \not\boxplus$ and $a \in V$.

– $mask^{\leftarrow}(\pi)$ returns the picture obtained from π by transforming all visible symbols from the leftmost column of the maximal visible subpicture of π into their invisible copies. Analogously, one defines the mappings $mask^{\rightarrow}$, $mask^{\uparrow}$, and $mask^{\downarrow}$.

– $unmask^{\leftarrow}(\pi)$ returns the picture obtained from π as follows. If ${}^{[i,j]}\pi_{[k,l]}$ is the maximal visible subpicture of π, then all invisible symbols $\pi(s, j-1)$, $i \leq s \leq k$, become visible. If $j = 1$, then $unmask^{\leftarrow}(\pi) = \pi$. Analogously, one defines the mappings $unmask^{\rightarrow}$, $unmask^{\uparrow}$, and $unmask^{\downarrow}$.
For every $\alpha \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ and $L \subseteq (V \cup \not\exists)^{**}$, we define $mask^{\alpha}(L) = \{mask^{\alpha}(\pi) \mid \pi \in L\}$. Analogously, $unmask^{\alpha}(L) = \{unmask^{\alpha}(\pi) \mid \pi \in L\}$.

For two disjoint subsets P and F of an alphabet V and a picture π over V, we consider the following two predicates which we will later use to define two types of filters:

$$rc_s(\pi; P, F) \equiv P \subseteq alph(\pi) \wedge F \cap alph(\pi) = \emptyset$$
$$rc_w(\pi; P, F) \equiv alph(\pi) \cap P \neq \emptyset \wedge F \cap alph(\pi) = \emptyset.$$

The construction of these predicates is based on *context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, both conditions require that no forbidding symbol is present in π; furthermore the first condition requires all permitting symbols to appear in π, while the second one requires that at least one permitting symbol appears in π.

For every picture language $L \subseteq V^{**}$ and $\beta \in \{s, w\}$, we define:

$$rc_{\beta}(L, P, F) = \{\pi \in L \mid rc_{\beta}(\pi; P, F) = \texttt{true}\}.$$

An *evolutionary picture processor* over $V \cup \not\exists$ is a 5-tuple (M, PI, FI, PO, FO), where:

– Either $M \subseteq Sub_V$ or $M \subseteq Del_V$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" into one type of evolutionary operation, only.

– $PI, FI \subseteq V$ are the *input* sets of permitting/forbidding symbols (contexts) of the processor, while $PO, FO \subseteq V$ are the *output* sets of permitting/forbidding symbols of the processor (with $PI \cap FI = \emptyset$ and $PO \cap FO = \emptyset$).

A *hiding picture processor* over $V \cup \not\exists$ is a 5-tuple (M, PI, FI, PO, FO), where M is either *mask* or *unmask*, while the other parameters are identical to those defined above for evolutionary processors.

An *accepting network of picture processors* (ANPP) is a 9-tuple

$$\Gamma = (V, U, G, N, \alpha, \beta, \underline{In}, \underline{Halt}, \underline{Accept}),$$

where:

– V and U are the input and network alphabet, respectively, $V \subseteq U$.
– $G = (X_G, E_G)$ is an undirected graph without loops with the set of vertices X_G and the set of edges E_G. G is called the *underlying graph* of the network. Although in network theory, several types of graphs are common like *complete, rings, stars, grids*, we focus here on complete underlying graphs (every two vertices are connected by an edge), so that we can replace the graph G by the set of its nodes.
– N is a mapping which associates with each node $x \in X_G$ the picture processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
– $\alpha : X_G \longrightarrow \{\leftarrow, \rightarrow, \uparrow, \downarrow, |, -, +\}$; $\alpha(x)$ gives the action mode of the rules of node x on the pictures existing in that node.
– $\beta : X_G \longrightarrow \{s, w\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

– input filter: $\rho_x(\cdot) = rc_{\beta(x)}(\cdot; PI_x, FI_x)$,
– output filter: $\tau_x(\cdot) = rc_{\beta(x)}(\cdot; PO_x, FO_x)$.
That is, $\rho_x(\pi)$ (resp. $\tau_x(\pi)$) indicates whether or not the picture π can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of pictures of L that can pass the input (resp. output) filter of x.

– $\underline{In}, \underline{Halt}, \underline{Accept} \in X_G$ are the *input* node, the *halting* node, and the *accepting* node of Γ, respectively. Note that it is not obligatory the three nodes be different from one another.

We say that $card(X_G)$ is the size of Γ. A *configuration* of an ANPP Γ as above is a mapping $C : X_G \longrightarrow 2^{U_*^*}$ which associates a finite set of pictures with every node of the graph. A configuration may be understood as the sets of pictures which are present in any node at a given moment. Given a picture $\pi \in V_*^*$, the initial configuration of Γ on π is defined by $C_0^{(\pi)}(\underline{In}) = \{\pi\}$ and $C_0^{(\pi)}(x) = \emptyset$ for all $x \in X_G - \{\underline{In}\}$.

A configuration can change via either a *processing step* or a *communication step*. When changing via a processing step, each component $C(x)$ of the configuration C is changed in accordance with the set of rules M_x associated with the node x and the way of applying these rules, namely $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one processing step* from the configuration C, written as $C \Longrightarrow C'$, iff $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_G$.

When changing via a communication step, each node processor $x \in X_G$ sends one copy of each picture it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the pictures sent by any node processor connected with x provided that they can pass its input filter.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = (C(x) \setminus \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))) \text{ for all } x \in X_G.$$

Note that pictures that cannot pass the output filter of a node remain in that node and can be further modified in the subsequent evolutionary steps, while pictures that can pass the output filter of a node are expelled. Further, all the expelled pictures that cannot pass the input filter of any node are lost.

Let Γ be an ANPP, the computation of Γ on an input picture $\pi \in V_*^*$ is a sequence of configurations $C_0^{(\pi)}, C_1^{(\pi)}, C_2^{(\pi)}, \ldots$, where $C_0^{(\pi)}$ is the initial configuration of Γ on π, $C_{2i}^{(\pi)} \Longrightarrow C_{2i+1}^{(\pi)}$ and $C_{2i+1}^{(\pi)} \vdash C_{2i+2}^{(\pi)}$, for all $i \geq 0$. Note that configurations are changed by alternative steps. By the previous definitions, each configuration $C_i^{(\pi)}$ is uniquely determined by $C_{i-1}^{(\pi)}$. A computation as above *halts* if there exists a configuration such that the sets of pictures existing in the halting node is non-empty. As we consider here ANPPs as problem solvers, for the rest of this paper we only deal with ANPPs that halt on every input. The *picture language decided* by Γ is

$L(\Gamma) = \{\pi \in V_*^* \mid$ the computation of Γ on π halts with a non-empty accepting node$\}$.

An ANPP without hiding picture processors is called accepting network of evolutionary picture processors (ANEPP) in [4]. The computational power of ANEPPs

has been investigated in [4]; we recall the following results, where the class of recognizable and local languages, respectively, have been defined in [7].

Theorem 1.
1. There exist non-recognizable languages which can be accepted by ANEPPs.
2. The complement of every local language can be accepted by an ANEPP.

3 Solving Picture Matching With ANPPs

A natural problem is to find a pattern (a fixed picture) in a given picture. This problem is widely known as the two-dimensional pattern matching problem and is largely motivated by different aspects in low-level image processing [13]. The more general problem of picture matching (it is not obligatory for the picture to be a two-dimensional array) is widely known in Pattern Recognition field and is connected with Image Analysis and Artificial Vision [11,20].

We discuss a solution to the problem of picture pattern matching based on the networks defined in the previous section. For the sake of a better understanding, we discuss first a solution based on ANEPP. A key step in our solution is to construct a network able to decide the singleton language formed by a given picture. If the given picture π is of size (k, n) or (n, k) for any $1 \leq k \leq 3$ and $n \geq 1$, then an ANEPP can decide the language $\{\pi\}$.

Theorem 2. *Let π be a picture of size (k, n) for some $1 \leq k \leq 3$ and $n \geq 1$. The language $\{\pi\}$ can be decided by an ANEPP.*

Proof. Actually, we only prove the most difficult case, namely $k = 3$, the proofs of the other cases that can be easily deduced from this one are left to the reader. We construct the ANEPP Γ deciding $\{\pi\}$ as follows. Let V be the alphabet of π; the working alphabet of Γ is:

$U = V \cup \{\overline{a^{(i)}}, \overline{a(i)}, \overline{a_{(i)}} \mid a \in V, 1 \leq i \leq n\} \cup \{[a, i] \mid a \in V, 1 \leq i \leq 4n\} \cup V'$
$V' = \{a' \mid a \in V\}$.

The nodes of Γ are distributed in four groups for a better understanding of their role.

Group 1.

$$\underline{In} : \begin{cases} M = \{\pi(1,1) \to \overline{\pi(1,1)^{(1)}}\}, \\ PI = V, FI = U \setminus V, \\ PO = \{\overline{\pi(1,1)^{(1)}}\}, FO = \emptyset, \\ \alpha = \uparrow,\ \beta = w. \end{cases}$$

Group 2.

$$x_{(i)} : \begin{cases} M = \{\pi(3,i) \to \overline{\pi(3,i)_{(i)}}\}, \\ PI = \{\overline{\pi(1,i)^{(i)}}, \overline{\pi(1,i+1)^{(i+1)}}\}, \\ FI = U \setminus (V \cup \{\overline{\pi(1,i)^{(i)}}, \overline{\pi(1,i+1)^{(i+1)}}\}), \\ PO = \{\overline{\pi(3,i)_{(i)}}\}, FO = \emptyset, \\ \alpha = \downarrow,\ \beta = s, \\ 1 \leq i \leq n-1 \end{cases}$$

$$x^{(i)}:\begin{cases} M=\{\pi(1,i)\to\overline{\pi(1,i)^{(i)}}\},\\ PI=\{\overline{\pi(1,i-1)^{(i-1)}}\},\\ FI=U\setminus(V\cup\{\overline{\pi(1,i-1)^{(i-1)}}\}),\\ PO=\{\overline{\pi(1,i)^{(i)}}\},FO=\emptyset,\\ \alpha=\uparrow,\ \beta=s,\\ 2\le i\le n \end{cases}\quad x(i):\begin{cases} M=\{\pi(2,i)\to\overline{\pi(2,i)(i)}\},\\ PI=\{\overline{\pi(1,i)^{(i)}},\overline{\pi(3,i)_{(i)}}\},\\ FI=\emptyset,\\ PO=\{\overline{\pi(2,i)(i)}\},\\ FO=\emptyset,\\ \alpha=\leftarrow,\ \beta=s,\\ 1\le i\le n \end{cases}$$

$$x_{del}^{(i)}:\begin{cases} M=\{\overline{\pi(1,i)^{(i)}}\to\varepsilon\},\\ PI=\{\overline{\pi(2,i)(i)}\},\\ FI=\emptyset,\\ PO=\emptyset,\\ FO=\{\overline{a^{(i)}},\overline{a_{(i)}},\overline{a(i)}\mid a\in V\},\\ \alpha=\leftarrow,\ \beta=s,\\ 1\le i\le n-1 \end{cases}\quad x_{(n)}:\begin{cases} M=\{\pi(3,n)\to\overline{\pi(3,n)_{(n)}}\},\\ PI=\{\overline{\pi(1,n)^{(n)}}\},\\ FI=U\setminus(V\cup\{\overline{\pi(1,n)^{(n)}}\}),\\ PO=\{\overline{\pi(3,n)_{(n)}}\},\\ FO=\emptyset,\\ \alpha=\downarrow,\ \beta=s. \end{cases}$$

Group 3.

Node	M	PI	FI	PO	FO	α	β
x_{err1}	$\{[a,i]\to[a,i+1]\mid a\in V,1\le i\le 4n-1\}$	$\{[a,1]\mid a\in V\}$	$\emptyset$	$\{[a,4n]\mid a\in V\}$	$\emptyset$	$+$	w
x_{err2}	$\{[a,4n]\to a'\mid a\in V\}\cup\{a\to a'\mid a\in V\}$	$\{[a,4n]\mid a\in V\}$	V'	U	$\{[a,4n]\mid a\in V\}$	$+$	w
x_{err}^{del1}	$\{a\to\varepsilon\mid a\in V\}$	V	$\{[a,4n]\mid a\in V\}$	U	$\emptyset$	$\leftarrow$	w
x_{err}^{del2}	$\{a\to\varepsilon\mid a\in V\}$	V	$\{[a,4n]\mid a\in V\}$	U	$\emptyset$	$\leftarrow$	w

The halting and the accepting node, which are grouped together, are defined respectively by

Group 4.

$$\underline{Halt}:\begin{cases} M=\emptyset,\\ PI=U\setminus V,\\ FI=V,\\ PO=\emptyset,\\ FO=\emptyset,\\ \alpha=*,\ \beta=s, \end{cases}\qquad \underline{Accept}:\begin{cases} M=\{\overline{\pi(1,n)^{(n)}}\to\overline{\pi(1,n)^{(n)}}\},\\ PI=\{\overline{\pi(1,n)^{(n)}},\overline{\pi(2,n)(n)},\overline{\pi(3,n)_{(n)}}\},\\ FI=V\setminus\{\overline{\pi(1,n)^{(n)}},\overline{\pi(2,n)(n)},\overline{\pi(3,n)_{(n)}}\},\\ PO=\emptyset,\\ FO=\{\overline{\pi(1,n)^{(n)}},\overline{\pi(2,n)(n)},\overline{\pi(3,n)_{(n)}}\},\\ \alpha=*,\ \beta=s, \end{cases}$$

It is easy to note that the nodes from Group 2 and those from Group 3 will never exchange pictures with each other. We analyze the computation of this network on an input picture μ of size (k,m) for some $k,m\ge 1$. In the input node $\underline{In}$, the following pictures are simultaneously produced: some pictures with $\overline{\pi(1,1)^{(1)}}$ on the first row, provided that $\pi(1,1)$ appears in the first row of μ, and several other pictures (at least one) all of them having exactly one symbol $[a,1]$ for some $a\in V$. We first assume that at least one picture with $\overline{\pi(1,1)^{(1)}}$ on its first row has been produced in $\underline{In}$ and follow the rest of the computation on such a picture. For simplicity we consider the case $k=3$ and $m=n$. All the pictures with $\overline{\pi(1,1)^{(1)}}$ on the first row which go out from $\underline{In}$ can be received only by either $x^{(2)}$, if $n\ge 2$, or $x_{(n)}$, if $n=1$. We assume $n\ge 2$ and continue

the computation in $x^{(2)}$. Here an occurrence of $\pi(1,2)$ on the first row of all pictures is replaced by $\overline{\pi(1,2)^{(2)}}$. All pictures where an occurrence of $\pi(1,2)$ has been replaced by $\overline{\pi(1,2)^{(2)}}$ can leave $x^{(2)}$ and enter $x_{(1)}$ where an occurrence of $\pi(3,1)$ on the last row is replaced by $\overline{\pi(3,1)_{(1)}}$. Now all pictures arrive in $x(1)$ where an occurrence of $\pi(2,1)$ on the leftmost column is replaced by $\overline{\pi(2,1)(1)}$. Note that if a picture does not have an occurrence of the symbol that is to be replaced in any of the nodes $x^{(2)}$, $x_{(1)}$, and $x(1)$, then it remains forever in that node.

Pictures going out from $x(1)$ can enter $x_{del}^{(1)}$ only, where the leftmost column is deleted provided that $\overline{\pi(1,1)^{(1)}}$ is situated on that column. The second condition to continue the computation is that $\overline{\pi(3,1)_{(1)}}$ is also situated on the column $\begin{matrix}\pi(1,1)\\ \cdots\cdots\\ \pi(2,1)\\ \cdots\cdots\\ \pi(3,1)\end{matrix}$ which is to be deleted in $x_{del}^{(1)}$. Therefore, the first column of μ must be .

Now the process described above resumes for all pictures going out from $x_{del}^{(1)}$, as all these pictures contain $\overline{\pi(1,2)^{(2)}}$ on their first row. Inductively, for every $1 \leq i \leq n-2$ every picture that has just gone out from $x_{del}^{(i)}$ must contain $\overline{\pi(1,i+1)^{(i+1)}}$ on its first row. Further on, it must follow the following itinerary through the network: $x^{(i+2)}$, $x_{(i+1)}$, $x(i+1)$, $x_{del}^{(i+1)}$.

We now analyze the case when the symbol on the first row of a picture going out from $x_{del}^{(n-1)}$ is $\overline{\pi(1,n)^{(n)}}$. This picture enters $x_{(n)}$ only, where an occurrence of $\pi(3,n)$ in the last row is replaced by $\overline{\pi(3,n)_{(n)}}$ and then enters $x(n)$ where an occurrence of $\pi(2,n)$ in the first column is replaced by $\overline{\pi(2,n)(n)}$. Now, if the picture is $\begin{matrix}\pi(1,n)^{(n)}\\ \pi(2,n)(n)\\ \pi(3,n)_{(n)}\end{matrix}$, then it enters simultaneously *Halt* and *Accept*, otherwise it is lost. By these explanations we infer the followings:

– If μ is of size $(3,n)$, then both nodes *Halt* and *Accept* become non-empty after $4n-1$ processing steps if and only if $\mu = \pi$.
– If $m < n$, then the computation on μ will be eventually blocked after at most $m-1$ column deletions.
– If $m > n$, then the computation on μ will be eventually blocked after at most $n-1$ column deletions.
– If $k < 3$, then the computation on μ is blocked after the first column deletion.
– If $k > 3$, then the computation on μ will be eventually blocked after at most $n-1$ column deletions.

We now analyze the computation on a picture containing a symbol $[a,1]$ which goes out from *In*. Such a picture enters x_{err1}, where $[a,1]$ is replaced successively by $[a,2]$, $[a,3]$, ..., $[a,4n]$. Hence, after $4n-1$ processing steps all pictures in x_{err1} contain a symbol $[a,4n]$, for some $a \in V$. They can leave now x_{err1} and enter x_{err2} only. A picture in x_{err2} can be transformed in two ways: a symbol

from V is replaced by its primed copy or $[a, 4n]$ is replaced by a'. In the former case, the picture cannot leave x_{err2}, while in the later, the picture leaves x_{err2} and one copy enters x^{del1} and another enters x^{del2}. In each of these nodes, the leftmost column is deleted provided that it contains a symbol from V. Now a "ping-pong" process starts between the two nodes x^{del1} and x^{del2}. This process continues until either the picture becomes empty or the leftmost column does not contain any symbol from V. If the current picture contains symbols from V' only, it enters $\underline{Halt}$ and the computation halts. By these explanations, we infer that $\underline{Halt}$ will always receive a picture from the nodes in Group 3 but not earlier than $4n$ processing steps.

In conclusion, the computation on μ always halts. It halts after either $4n-1$ processing steps, which means that $\mu = \pi$, or $4n-1+k'+n'-1 > 4n-1$ processing steps, provided that the input picture is of size (k', n'), hence $\mu \neq \pi$. □

We give now a solution to the picture matching based on ANEPP, provided that the pattern is of size (k, n) or (n, k) for any $1 \leq k \leq 3$ and $n \geq 1$.

Theorem 3. *Let π be a picture of size (k, l) for some $1 \leq k \leq 3$ and $l \geq 1$. The language $\{\theta \mid \pi$ is a subpicture of $\theta\}$ can be decided by an ANEPP.*

Proof. We give only an informal description of the construction which is based on a pretty simple idea. The network defined in the proof of Theorem 2 will be used as a subnetwork as follows. The node $\underline{In}$ of that network is renamed x_I all the other nodes remaining unchanged. The network we intend to construct contains nine nodes more:
– $\underline{In}$, which is a substitution node where a symbol is replaced by itself everywhere in the picture.
– two identical nodes deleting the leftmost column.
– two identical nodes deleting the rightmost column.
– two identical nodes deleting the uppermost row.
– two identical nodes deleting the undermost row.
All these nodes can receive pictures containing original symbols only such that as soon as a picture entered one node from any of the Groups 1,2,3, it cannot further returns to these nodes. As one can see, these new 9 nodes cut from the input picture arbitrary subpictures. Clearly, all subpictures of the same size are produced simultaneously. All subpictures of the same size received by the subnetwork are matched against the pattern π in parallel. A short discussion is in order here. Assume that an input picture is of size (m, n); all pictures of the same size (k', l') will be sent to the subnetwork after exactly $(m-k')+(n-l')+1$ processing steps. If at least one of these subpictures is identical to π, both halting and accepting node will eventually be non-empty after $m-k+n-l+4l$ processing steps. In this case, the input picture is accepted. If the halting node is empty after $m-k+n-l+4l$, it will definitely become non-empty after $m+n+4l-1$ processing steps. As $m+n+4l-1 > m-k+n-l+4l$, the input picture is rejected. □

Note that the network constructed in the previous proof (nodes, rules, filters, symbols) does not depend on the input picture but on the pattern only.

Various algorithms exist for the exact two-dimensional matching problem. The fastest algorithms for finding a rectangular picture pattern of size (k, l) in a given picture of size (n, m) run in $\mathcal{O}(n \times m + k \times l)$ time, see, e.g., [3,21]. It is rather easy to note that an ANEPP which decides whether a pattern of size (k, l), $1 \leq k \leq 3$, $l \geq 1$, appears in a given picture of size (n, m) does this in $\mathcal{O}(n+m+l)$ computational (processing and communication) steps. On the other hand, the space complexity of the algorithm proposed in [21] is $\mathcal{O}(n \times m + k \times l)$, while in our case the number of pictures moving through the network is exponential. We recall that some biological phenomena are sources of inspiration for our model. In this context, it is considered to be biologically feasible to have sufficiently many identical copies of a molecule. By techniques of genetic engineering, in a linear number of laboratory operations one can get an exponential number of identical 2-dimensional molecules [1,2].

It is worth mentioning that the construction described above can be easily extended to an ANEPP able to detect, in the same number of computational steps, any pattern from a finite sets of pictures of the same size. It suffices to construct an independent subnetwork for each pattern.

Theorem 4. *Given a finite set F of patterns of size (k, l) and (l, k) for all $1 \leq k \leq 3$ and $l \geq 1$, the pattern matching problem with patterns from F can be solved by ANEPPs in $\mathcal{O}(n+m+l)$ computational (processing and communication) steps.*

However, this approach is not suitable for detecting patterns of a different size that those considered above. In the sequel, we show how the picture pattern matching can be completely solved with ANPP, that is with networks having both types of nodes: evolutionary processors and hiding processors. As the idea is the same, it suffices to construct an ANPP able to decide the singleton language formed by a given picture.

Theorem 5. *Let π be a picture of size (k, l), for some $k, l \geq 1$ over an alphabet V. The language $\{\pi\}$ can be decided by an ANPP.*

The idea of the proof is the same as that from the proof of Theorem 2, namely it consists in two disjoint subnetworks, one of them checking whether the input picture is identical to π, and the other one making a sufficiently long computation which ends in the halting node but allows the first network to complete its computation. The complete proof is left to the reader.

We are now able to give the complete solution based on ANPPs to the problem of picture matching:

Theorem 6. *Given a finite set F of patterns of size (k, l) and (l, k) for any $k, l \geq 1$, the pattern matching problem with patterns from F can be solved by ANPPs in $\mathcal{O}(n + m + kl + k)$ computational (processing and communication) steps.*

Clearly, the networks including both evolutionary and hiding processors seem to be more powerful than ANEPPs considered in [4]. A natural further step is to investigate the computational power and other computational properties of ANPPs.

References

1. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.: Running time and program size for self-assembled squares. In: Proc. 33rd ACM STOC, pp. 740–748 (2001)
2. Aggarwal, G., et al.: Complexities for generalized models of self-assembly. SIAM Journal on Computing 34, 1493–1515 (2005)
3. Amir, A., Benson, G., Farach, M.: Alphabet independent two dimensional matching. In: Proc. 24th ACM STOC, pp. 59–68 (1992)
4. Bottoni, P., Labella, A., Mitrana, V.: Networks of evolutionary picture processors. Fundamenta Informaticae 131, 337–349 (2014)
5. Bozapalidis, S., Grammatikopoulou, A.: Recognizable picture series. J. of Automata, Languages and Combinatorics 10, 159–183 (2005)
6. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Handbook of Formal Languages, pp. 215–267. Springer (1997)
7. Giammarresi, D., Restivo, A.: Recognizable picture languages. Int. J. Pattern Recognition and Artificial Intelligence 6, 241–256 (1992)
8. Inoue, I., Takanami, I.: A survey of two-dimensional automata theory. In: Dassow, J., Kelemen, J. (eds.) IMYCS 1988. LNCS, vol. 381, pp. 72–91. Springer, Heidelberg (1989)
9. Manea, F., Martín-Vide, C., Mitrana, V.: Accepting networks of evolutionary word and picture processors: A survey. In: Scientific Applications of Language Methods. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory, vol. 2, pp. 523–560. World Scientific (2010)
10. Margenstern, M., Mitrana, V., Jesús Pérez-Jímenez, M.: Accepting Hybrid Networks of Evolutionary Processors. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 235–246. Springer, Heidelberg (2005)
11. Marriott, K., Meyer, B.E.: Visual Language Theory. Springer (1998)
12. Maürer, I.: Characterizations of recognizable picture series. Theoretical Computer Science 374, 214–228 (2007)
13. Rosenfeld, A., Kak, A.C.: Digital Picture Processing. Academic Press, New York (1982)
14. Rosenfeld, A., Siromoney, R.: Picture languages – a survey. Languages of Design 1, 229–245 (1993)
15. Siromoney, G., Siromoney, R., Krithivasan, K.: Abstract families of matrices and picture languages. Computer Graphics and Image Processing 1, 284–307 (1972)
16. Siromoney, G., Siromoney, R., Krithivasan, K.: Picture languages with array rewriting rules. Information and Control 22, 447–470 (1973)
17. Subramanian, K.G., Siromoney, R.: On array grammars and languages. Cybernetics and Systems 18, 77–98 (1987)
18. Wang, P.S.: Hierarchical structure and complexities of parallel isometric patterns. IEEE Trans. PAM I 5, 92–99 (1983)
19. Wang, P.S.: Sequential/parallel matrix array languages. Journal of Cybernetics 5, 19–36 (1975)
20. Wang, P.S., Bunke, H. (eds.): Handbook on Optical Character Recognition and Document Image Analysis. World Scientific (1996)
21. Zhu, R.F., Takaoka, T.: A technique for two-dimensional pattern matching. Communications of the ACM 32, 1110–1120 (1989)